

```

cSHELL99cSHELL99
  cSHELL99      cSHELL99
    cSHELL99      cSHELL99
      cSHELL99      cSHELL99
        cSHELL99      cSHELL99
          cSHELL99      cSHELL99
            cSHELL99      cSHELL99
              cSHELL99      cSHELL99
                cSHELL99      cSHELL99
                  cSHELL99      cSHELL99
                    cSHELL99      cSHELL99
                      cSHELL99      cSHELL99
                        cSHELL99      cSHELL99
                          cSHELL99      cSHELL99
                            cSHELL99      cSHELL99

```

```

SSSS  H   H  EEEEE  L       L
S      S   H   H   E       L       L       9999       9999
S      S   H   H   E       L       L       9   9       9   9
SSSS  HHHHHH EEEE   L       L       9999       9999
S      S   H   H   E       L       L       9       9
S      S   H   H   E       L       L       9       9
SSSS  H   H  EEEEE  LLLLLL LLLLLL       9       9

```

```

cSHELL99
cSHELL99
cSHELL99
cSHELL99
  cSHELL99
    cSHELL99
      cSHELL99
        cSHELL99
          cSHELL99
            cSHELL99
              cSHELL99
                cSHELL99
                  cSHELL99
                    cSHELL99
                      cSHELL99
                        cSHELL99
                          cSHELL99
                            cSHELL99
                              cSHELL99
                                cSHELL99
                                  cSHELL99
                                    cSHELL99
                                      cSHELL99
                                        cSHELL99
                                          cSHELL99
                                            cSHELL99
                                              cSHELL99
                                                cSHELL99
                                                  cSHELL99
                                                    cSHELL99

```

System	Disk Special
Work disk	
Printer	
Color	
Inf	
Mem	available = 160
Blk Background	#M16
Sav	#M12
Install	#M13
Remove	#M14
Single Dsk	#M15
	#M16
	#M17
	#M18
	#M19
	#M20
	#M21
	#M22
	#M23
	#M24
	#M25
	#M26
	#M27
	#M28
	#M29
	#M30
	#M31
	#M32
	#M33
	#M34
	#M35
	#M36
	#M37
	#M38
	#M39
	#M40
	#M41
	#M42
	#M43
	#M44
	#M45
	#M46
	#M47
	#M48
	#M49
	#M50
	#M51
	#M52
	#M53
	#M54
	#M55
	#M56
	#M57
	#M58
	#M59
	#M60
	#M61
	#M62
	#M63
	#M64
	#M65
	#M66
	#M67
	#M68
	#M69
	#M70
	#M71
	#M72
	#M73
	#M74
	#M75
	#M76
	#M77
	#M78
	#M79
	#M80
	#M81
	#M82
	#M83
	#M84
	#M85
	#M86
	#M87
	#M88
	#M89
	#M90
	#M91
	#M92
	#M93
	#M94
	#M95
	#M96
	#M97
	#M98
	#M99
	#M100
	#M101
	#M102
	#M103
	#M104
	#M105
	#M106
	#M107
	#M108
	#M109
	#M110
	#M111
	#M112
	#M113
	#M114
	#M115
	#M116
	#M117
	#M118
	#M119
	#M120
	#M121
	#M122
	#M123
	#M124
	#M125
	#M126
	#M127
	#M128
	#M129
	#M130
	#M131
	#M132
	#M133
	#M134
	#M135
	#M136
	#M137
	#M138
	#M139
	#M140
	#M141
	#M142
	#M143
	#M144
	#M145
	#M146
	#M147
	#M148
	#M149
	#M150
	#M151
	#M152
	#M153
	#M154
	#M155
	#M156
	#M157
	#M158
	#M159
	#M160
	#M161
	#M162
	#M163
	#M164
	#M165
	#M166
	#M167
	#M168
	#M169
	#M170
	#M171
	#M172
	#M173
	#M174
	#M175
	#M176
	#M177
	#M178
	#M179
	#M180
	#M181
	#M182
	#M183
	#M184
	#M185
	#M186
	#M187
	#M188
	#M189
	#M190
	#M191
	#M192
	#M193
	#M194
	#M195
	#M196
	#M197
	#M198
	#M199
	#M200
	#M201
	#M202
	#M203
	#M204
	#M205
	#M206
	#M207
	#M208
	#M209
	#M210
	#M211
	#M212
	#M213
	#M214
	#M215
	#M216
	#M217
	#M218
	#M219
	#M220
	#M221
	#M222
	#M223
	#M224
	#M225
	#M226
	#M227
	#M228
	#M229
	#M230
	#M231
	#M232
	#M233
	#M234
	#M235
	#M236
	#M237
	#M238
	#M239
	#M240
	#M241
	#M242
	#M243
	#M244
	#M245
	#M246
	#M247
	#M248
	#M249
	#M250
	#M251
	#M252
	#M253
	#M254
	#M255
	#M256
	#M257
	#M258
	#M259
	#M260
	#M261
	#M262
	#M263
	#M264
	#M265
	#M266
	#M267
	#M268
	#M269
	#M270
	#M271
	#M272
	#M273
	#M274
	#M275
	#M276
	#M277
	#M278
	#M279
	#M280
	#M281
	#M282
	#M283
	#M284
	#M285
	#M286
	#M287
	#M288
	#M289
	#M290
	#M291
	#M292
	#M293
	#M294
	#M295
	#M296
	#M297
	#M298
	#M299
	#M300
	#M301
	#M302
	#M303
	#M304
	#M305
	#M306
	#M307
	#M308
	#M309
	#M310
	#M311
	#M312
	#M313
	#M314
	#M315
	#M316
	#M317
	#M318
	#M319
	#M320
	#M321
	#M322
	#M323
	#M324
	#M325
	#M326
	#M327
	#M328
	#M329
	#M330
	#M331
	#M332
	#M333
	#M334
	#M335
	#M336
	#M337
	#M338
	#M339
	#M340
	#M341
	#M342
	#M343
	#M344
	#M345
	#M346
	#M347
	#M348
	#M349
	#M350
	#M351
	#M352
	#M353
	#M354
	#M355
	#M356
	#M357
	#M358
	#M359
	#M360
	#M361
	#M362
	#M363
	#M364
	#M365
	#M366
	#M367
	#M368
	#M369
	#M370
	#M371
	#M372
	#M373
	#M374
	#M375
	#M376
	#M377
	#M378
	#M379
	#M380
	#M381
	#M382
	#M383
	#M384
	#M385
	#M386
	#M387
	#M388
	#M389
	#M390
	#M391
	#M392
	#M393
	#M394
	#M395
	#M396
	#M397
	#M398
	#M399
	#M400
	#M401
	#M402
	#M403
	#M404
	#M405
	#M406
	#M407
	#M408
	#M409
	#M410
	#M411
	#M412
	#M413
	#M414
	#M415
	#M416
	#M417
	#M418
	#M419
	#M420
	#M421
	#M422
	#M423
	#M424
	#M425
	#M426
	#M427
	#M428
	#M429
	#M430
	#M431
	#M432
	#M433
	#M434
	#M435
	#M436
	#M437
	#M438
	#M439
	#M440
	#M441
	#M442
	#M443
	#M444
	#M445
	#M446
	#M447
	#M448
	#M449
	#M450
	#M451
	#M452
	#M453
	#M454
	#M455
	#M456
	#M457
	#M458
	#M459
	#M460
	#M461
	#M462
	#M463
	#M464
	#M465
	#M466
	#M467
	#M468
	#M469
	#M470
	#M471
	#M472
	#M473
	#M474
	#M475
	#M476
	#M477
	#M478
	#M479
	#M480
	#M481
	#M482
	#M483
	#M484
	#M485
	#M486
	#M487
	#M488
	#M489
	#M490
	#M491
	#M492
	#M493
	#M494
	#M495
	#M496
	#M497
	#M498
	#M499
	#M500
	#M501
	#M502
	#M503
	#M504
	#M505
	#M506
	#M507
	#M508
	#M509
	#M510
	#M511
	#M512
	#M513
	#M514
	#M515
	#M516
	#M517
	#M518
	#M519
	#M520
	#M521
	#M522
	#M523
	#M524
	#M525
	#M526
	#M527
	#M528
	#M529
	#M530
	#M531
	#M532
	#M533
	#M534
	#M535
	#M536
	#M537
	#M538

DISCLAIMER

Joseph Ross, the sole author and distributor of cSHELL99, referred to as the "product", does not guarantee that this product or manual will be free from error, or meet the needs or expectations of the user.

The author is not liable for the use or misuse of this product or any damage that may result from the normal use of this program, proper or improper use of this product or any information contained within this manual.

The author warranties the product diskette for a period not to exceed 90 days from the purchase date provided the diskette is not damaged from improper use, accident, or any condition not due to the quality of the material or workmanship. The author reserves the right to refuse to service any returned materials that do not meet this qualification.

If there is a diskette problem that meets the above conditions, send the original diskette only to the author and a free replacement will be issued to the user.

Because of the nature of the cSHELL99 product, the product is provided without any copy protection. This is not a license to distribute this product. The product and manual are copyrighted by the author and may not be reproduced by any means for the use of others. If the user transfers ownership to another, all personal copies must be transferred or destroyed. It is the owners responsibility to control its use and distribution of personal copies. Owners who disregard this responsibility are liable for any damages incurred to the author due to the loss of sales.

Support for cSHELL99 will be in direct proportion to the response for the product. All comments and product suggestions are welcome and should be forwarded by mail to the author. In the near future I plan to create numerous companion modules that will be offered at a very reasonable price. Also in the works is two versions of the product that will take advantage of the Super Space & Super SpaceII memory cartridge by DataBiotics Inc.

MANUAL: Copyright 1989 - Joseph Ross
PROGRAM: Copyright 1989 - Joseph Ross
PRODUCED BY:

Joseph Ross
119 Knollwood Terrace
Clifton, New Jersey 07012

ALL RIGHTS RESERVED.
Purchase price \$30.00 US funds.

Copies of the "c99" compiler may be purchased from:

Clint Pulley
38 Townsend Avenue
Burlington, Ontario
Canada L7T 1Y6

At last information the purchase price was \$20.00 .

FORWARD

cSHELL99 is a window oriented DOS type program similar in appearance to "GEOS" for the Commodore 64. It was developed as a personal project of the author for use on the TI 99/4A Home Computer. Unlike GEOS and other MacIntosh type systems, cSHELL99 does not operate in a bit mapped mode but rather in text mode. This was done to utilize the VDP RAM for window displaying routines and enable the use of various loaders which are vital to the program. It also contributes to the speed of the windows and greatly reduces the overhead needed to run the system.

cSHELL99 is written in a mix of assembler and "c99" created by Clint Pulley of Ontario, Canada with all of the assembly language routines being placed within a C structure. Some of the routines will have their source code included so that the user may take full advantage of and expand the basic system.

The purpose of cSHELL99 is to provide the 99/4A community with a slightly different and fun environment in which to operate. It is not intended to replace the many fine pieces of software already in the community but to demonstrate additional power of a great machine. In the main program, DESKTOP, there are many file and disk oriented features along with system configuration windows and program loaders. The main power of the system shell is the use of the loaders which allow programs written in "c99" to be run from the shell, execute and return back to the shell environment thus enabling the user to add features they desire. The documentation provided describes in-depth the system's features, use of loaders, "c" stack, REF/DEF table, system functions and globals available to the user. Interfacing with cSHELL99 is a rather easy process for both experienced and beginning "C" programmers as you will see when you read the documentation. Simple examples of interfacing have been provided to aid in this process. The loaders used are the two standard loaders used in the Ed/Assm environment, option #3 Load and Run and option #5 Program Image.

Also included with the cSHELL99 package is a c99 sound library which is similar to the Basic CALL SOUND command, The source code to a c99 graphics library, a c99 speech library and many modules that demonstrate the ability of c99 to execute text, graphics and bitmap modes.

I believe you will find cSHELL99 a most interesting and enjoyable environment in which to operate. With the tools and information provided, you will soon discover that the limits of the TI 99/4A are not 32K but your imagination.

Enjoy,

Joe Ross

LOADING cSHELL99

The equipment needed to run cSHELL99 is as follows:

- TI 99/4A Home Computer
- PE Box or the equivalent
- 32K memory expansion
- Disk controller and disk drive
- Editor Assembler module, Extended Basic or
- TI Writer module

Optional but desirable:

- Additional disk drives
- RS232 card
- Printer
- Joystick #1 or trackball

If you have received cSHELL99 on SS/SD floppy disks, place the SUSTEM disk side A in a drive and follow the loading procedure. When the program has loaded, turn the system disk to side B to access the desktop modules.

LOADING

To run the program, select option #5 from the Editor Assembler main menu and type DSK#.UTIL1 or just press ENTER if the system disk is in drive #1.

From Funlwriter select Loaders and choose the Program loader. Type DSK#.UTIL5 and the program will load and start running in the Desktop screen.

To load from the Extended Basic cartridge, choose Extended Basic from the selection screen and the program will automatically load and run if the system disk is in drive #1. Else type RUN DSK#.LOAD .

To load from the TI Writer module, select Utilities and press ENTER if the system disk is in drive #1 or type DSK#.UTIL1.

NOTE: The Alpha/Lock key must be in the UP position if using the joystick.

cSHELL99

Desktop.....	A.1
Module Auto Running.....	B.1
Linking Loader.....	C.1
Creating Program Image Files.....	D.1
cSHELL99 System Functions.....	E.1
cSHELL99 Window Functions.....	F.1
Window Globals.....	G.1
Companion Modules.....	H.1
Screen Dump Module.....	I.1
Sound Library.....	J.1
Speech Library.....	K.1
Low Memory Usage.....	AP.1
VDP Memory Map.....	AP.2
FILE I/O Errors.....	AP.3
System Globals.....	AP.4
c99 Stack.....	AP.5
Desktop Character Definitions.....	AP.6

Added Documentation

Source code and additional information on MODULES DISK
sides B and D.

Desktop Symbols

-  Open automode disk
-  Close automode disk
-  Trash - delete file
-  Printer - print text file
-  Auto catalog pager

System File Disk Special

Desktop Windows Access

Used = 594 Available = 124

AUTOMODE;C	d	DRENAME;C	d
BDUMP;C	dd	EARTH;C	dd
BOOT;S	dd	FRENAME;C	dd
CCAT;C	dd	CCTCOH;C	dd
CHANGE2;S	dd	LINK2;C	dd
CHANGEIT;C	dd	LIST2;C	dd
COLOR;C	dd	MSIZE;C	dd
COUNT;C	dd	PRINT;C	dd
DESKTOP;C	dd	PRNAME;C	dd
DESKTOP;0	d	PROTUN;C	d

Automode Catalog

System
Work Disk
Printer
Color
Blank Scrn
Into File
Memory
Save
Install
Remove
Single Dsk

1

File
View
Copy
Rename
Print
Search
Text Size
Delete
Protect
Unprotect

2

DISK
Catalog
Rename
Back-Up

3

Special
Load & Run
Link & Run
Batch Run
Program Ldr.
Editor
c99 Compiler
Exit cSHELL99

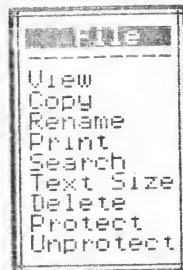
4

Desktop Windows

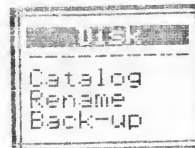
cSHELL99 System Windows



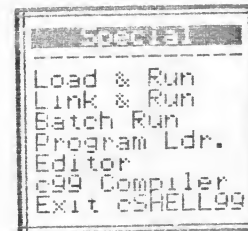
#1



#2



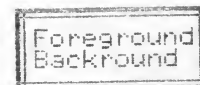
#3



#4



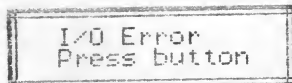
#5



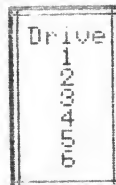
#6



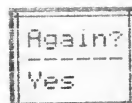
#7



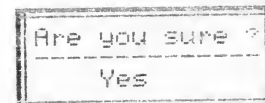
#8



#9



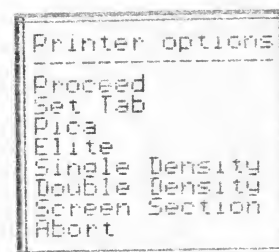
#10



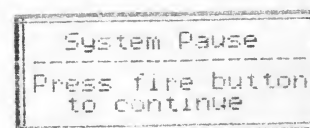
#11



#14



#12



#13

Blank windows are multiple purpose windows.

DESKTOP

cSHELL99 starts in what will be known as the DESKTOP environment. From the DESKTOP the user may access all the program modules and system utilities via dialog boxes (windows) and icons. To activate any icon or dialog box, place the pointer (arrow) on top of the desired selection by using joystick #1 and CLICK (press the fire button on the joystick). If a dialog box is activated, a window will appear displaying a menu. To select an item from the menu, pull or push the joystick forward or backward till the desired selection appears in inverse video, then CLICK once. The result of selecting an item from a window menu will be to execute the selection, prompt for input or display another window menu. If no item from the window menu is desired, move the joystick till the inverse video disappears and CLICK. The window will close leaving the pointer in its previous position. When an icon (pictorial representation) is activated a predetermined action or message is the result.

NOTE: The arrow keys may be substituted for the action of the joystick in both moving the pointer and choosing a selection from a dialog box menu. The "Q" key may be used instead of the joystick fire button.

*Mouse ?
button*

DESKTOP WINDOWS/ICON ACCESS

The desktop windows are accessed by placing the pointer on the desired title (System,File,Disk,Special) and pressing the fire button on the joystick. A window will open on the screen as a menu ready for your selection. Pull/push the joystick untill the desired option appears in inverse video and click once. If the result of the selection is another window menu, select the secondary selection in the same manner. Many of the desktop options require a filename to perform their process. There are two ways to supply the option with the filename, auto processing mode and prompts. The desktop windows options, printer icon and trash can icon will check whether a file or program module has been selected from the auto catalog on the desktop screen. If an auto processing catalog is present and a filename has been selected, the filename will be passed to the desktop window option or icon and be automatically processed. If no filename is selected for auto processing, you will be prompted for a device and filename. Example: DSK#.FILENAME . Using the prompt method allows for random access to any file in any valid device. Using the auto processing mode method allows for ease of use and speed in getting around the desktop options.

AUTO PROCESSING MODE

To initially activate the auto processing mode of the desktop, place the pointer on the word "System" of the desktop screen and press the fire button. When the system menu window appears, select "Work Disk" and click once. Another window menu will appear displaying disk drive numbers. Select the drive number you desire to be auto cataloged and click the fire button. All windows will disappear and the drive number selected will be placed to the upper right of the auto catalog section.

To catalog a disk and make its files available for the automatic processing mode, place the pointer on the disk icon and click the fire button once. The auto catalog module will load and run from the system disk and the first page of filenames from the selected disk will be displayed on the desktop screen in the auto catalog section. To view additional filenames, place the pointer on the pager icon (The box that looks like a bent page corner located below and to the left of the file names) and click the fire button once. With each click the next page of filenames will be displayed. The paging of the auto catalog is circular with the first page following the last. A file/program name may be selected for auto processing only from the page currently on the screen. Clicking on the pager icon or closing the auto catalog with the disk close button, will deactivate the previously selected file from being auto processed. To select a file for auto processing, place the pointer on the filename and press the fire button. The filename will appear in inverse video and will be available to the desktop options. To select another file for auto processing, the previous file must be turned off by the same manner then another file may be chosen.

If you wish to auto catalog another disk, place the pointer on the disk close button and press the fire button once. This will clear the auto catalog on the desktop screen. To catalog the new disk place the pointer on the disk icon and click the fire button. If the new disk is in a disk drive other than the one that is displayed to the right of the auto catalog section, you must change the drive number from the "Work Disk" window menu then proceed to access the disk icon.

DESKTOP WINDOWS

System

Work Disk

Selecting this item will cause another window menu to appear on the screen. The disk drive chosen from this menu will be the disk opened to the auto driven mode when the disk icon is clicked. You will notice that the "1" to the upper right will now display the number of the drive selected (Drive one is the default drive) . This selection may be accessed only when no auto catalog is opened and displayed. Once a disk has been auto cataloged, the Close Disk button must be clicked, then access to this option will again be available.

Printer

Choosing this item will cause a system module to load and display another window with the current printer name. If the name must be changed, select the change name line and click. Type the new printer name and press ENTER. The name designated in this option is the device to which files will be printed.

Color

The selection of Color will cause the color change module to load and run from the system disk and display another window for selecting a change in either the foreground or background color. Choose the desired option and click once. Now you may change the selected foreground/background color by moving the joystick. When the color is on the screen click once and you may again make a selection. If no further changes are needed pull/push the joystick till no selection is in inverse video and click once.

Info

Selecting Info will cause the file listing module to load, run and display an information file. As in viewing a file, you may page through the file by pushing the joystick forward or left and right to see other sections of the current page. The file used is "INFO" on the system disk. Feel free to edit this file with any text editor and add any information you wish. When you wish to quit viewing the file, click the fire button once.

Blank Scrn

This option will cause the monitor screen to blank out. To resume, press the fire button or "Q" on the keyboard and the desktop screen will again be visible.

Memory

The memory option will load and run a system status module which will display the current system configuration. The information displayed is as follows:

1. High memory - The number of bytes left in high memory to load and run program modules (Ed/Assm option #3 & 5) written in c99.
2. Low memory - The number of bytes left for "c99" stack usage before an overlap into the Ed/Assm utilities would occur. This is also the same memory area left for loading program modules into low memory. If stack usage is minimal, program modules may load here without adverse affects.
3. Stack pointer - The current decimal location of the c99 stack.
4. REF/DEF pointer - The current decimal location of the bottom of the REF/DEF table.
5. REF/DEF table - the number of bytes available to the REF/DEF table for additional labels before an overlap into the c99 stack. The number of available labels is this number divided by 8 bytes per entry. Upon exit of a user module its label space used is reclaimed and made available to the next user created program.
6. No. windows defined - This is the number of windows already created and active in cSHELL99. These windows may be used in user created programs. Also includes user created windows. (See cSHELL99 Windows)
7. Windows available - The number of user defined windows available for creation. (See cSHELL99 WINDOWS)
8. No. window bytes used - This is the total number of bytes used in VDP RAM for currently defined windows.
9. No. window bytes available - The number of bytes available for user defined windows. The number of bytes for a particular window is the number of characters in width times the number of lines that the window occupies. Example: A window 20 characters wide and 5 lines in length would take 100 bytes of VDP RAM. This amount of memory available for window definition will vary depending on the number of files in the current auto catalog if one is open. (See VDP MEMORY MAP)
10. Mutiple/Single Drive System - This message displays whether you have chosen to run as a multiple or single drive system. (See Single Dsk under the System desktop window)

Save

Save will write the screen color information and the printer name to a system file as a permanent change to the program so the user need not configure the program with each rerun. Make sure that the system disk with the file \$CONFI (SYSTEM DISK Side A) is in a drive which can be accessed. I strongly suggest that you copy the system disk with your favorite copy program and work from the copy disk.

Install

*Do they have to be
reinstalled
everything if it's
used?*

Install will allow the user to add c99 libraries to the cSHELL99 shell as if it were part of the original program. The installed libraries and related labels may be accessed by programs loaded with the loaders without having to load them in each time they are needed (as with Link & Run). Any normal c99 library may be installed without modification as long as there isn't a double definition error caused by an entry label. To check the memory available after the installation of a library, choose Memory from the System window and the updated information for relocatable modules will be displayed. If an error occurs during the installation of a module, an error window with the I/O error number will be displayed. Press the fire button or "Q" to remove the error window.

Remove

Remove will delete all installed libraries and reset the memory pointers for the cSHELL99 environment to their original values.

Single Dsk

This option will toggle between a single or multiple disk system. Many of cSHELL99's desktop window options load an option module from the system disk which in turn accesses the file and disk selected for processing. By selecting to run as a single disk system, the desktop program modules after loading will pause and display a pause window. If necessary, swap disks at this time and press the fire button to continue.

File

View

View will load and run the file listing module from the system disk and display the selected text file in a window. To page through a file push the joystick forward to advance to the next page. To view other sections of the current page (As in TI Writer or ED/ASSM Editor) move the joystick left or right. Below the viewing window is another window showing the line number of the last line in the viewing window and the starting column number of the page section being viewed. To quit viewing a file press the fire button once.

Copy

Copy will load and run a file copying module from the system disk and prompt for two filenames, the original filename and the copy filename. Example: DSK1.ORIGINAL and DSK2.COPY. (In auto mode the original filename is supplied automatically). To abort this task press ENTER without supplying a name. As of now the file copying module will only work with two drives unless you are copying a file to a different name on the same disk.

Rename

The file renaming module will load from the system disk and prompt for a disk #, old filename and new filename. Example: 1, OLDNAME, NEWNAME. (In auto mode the disk drive number and old filename is supplied) It will then rename the file as requested and return to the desktop. To abort this process just press ENTER when prompted for a filename.

Print & Printer Icon

The text printing module will load from the system disk and prompt for a device and filename if no selection was made from the auto catalog. Example: DSK#.Fname . If the text file is found, a window will appear on the screen in menu mode and allow the user to set the lines that will be printed if desired. To set the printing range, choose "Set parameters" from this window and you will be prompted for the line numbers to start and stop the printing. Enter the information and the lines between the start line and stop line (inclusive) or end of file will be printed. To print the entire file, press the fire button without selecting Set parameters. The default values are start line = one and the stop line = 10000 or end of file. Once the printing has started, it may be stopped by pressing the fire button. This selection may also be used to create a text file from any section of an existing text file by changing the printer name to a disk filename before accessing this module. If you do this, remember to reset the printer name when finished.

Search

The Search module will search a display variable/fixed file for any key up to 80 characters. When a record containing the key is found, the record number and the entire record will be displayed on the screen. To pause the search press the space bar and to resume release it. DO NOT press FCNT/4 or the program will abort. When the entire file has been searched, a message will be displayed. Press the fire button to exit the module and return to desktop. If a file has been selected for auto processing the filename will be supplied else at the prompt enter DSK#.FILENAME .

NOTE: The search is case sensitive.

Text Size

This module will display the number of lines and characters in D/V 80 file. After the file data is displayed, press the fire button to exit. In auto mode the filename will be supplied else at the prompt enter DSK#.FILENAME.

Delete & Trash Can Icon

If in auto mode the filename will be supplied else enter DSK#.FILENAME of the file you wish to delete. You then will be asked if you are sure you want to delete the file. Press "Y" to delete the file or abort the request by pressing any other key. If a file is protected it will not be deleted and no error message will be issued.

Protect

Choosing this option will load and run a file protection module. At the prompts enter the disk number and the filename you wish to protect. If in auto mode, the filename will be supplied and automatically processed. The file will be protected and the module will exit to cSHELL99. If the file can not be found on the specified disk, a file I/O message will be shown. Click the fire button once to exit the module.

Unprotect

The unprotect option will remove the file protection from a file if it is protected otherwise no action will be taken. At the prompts enter the disk number and filename of the file you wish to unprotect. In auto mode the disk and filename will be provided for you. The file will be unprotected and exit to desktop. If the file can not be found on the specified device, a file I/O window will be placed on the screen. Click the fire button to remove the window and exit to desktop.

DISK

Catalog

A traditional disk catalog program written by Clint Pulley and disassembled by Tom Wible will load and display a window with disk drive numbers. Select the desired drive and a catalog of the selected disk will be shown. If no catalog is desired, press the fire button without making a selection and you will exit the module. When prompted to press a key, you may press any key on the left side of the keyboard or the fire button. The file symbols used are the same as in the auto catalog and are as follows:

- d = Display variable (text)
- D = Display fixed (ED/ASSM option #3 programs)
- i = Internal variable
- I = Internal fixed
- P = Program Image (Basic programs, ED/ASSM option #5 programs or binary data files)

Rename

The disk renaming module will load and run and prompt for a disk #. The selected disk will then have its name displayed and you will be asked for a new name. After the new name is entered, the selected disk will be renamed as requested and the module will exit. To abort this process enter zero for the disk number. DO NOT rename the System or Modules disks or the modules will not load properly as intended. The sytem disk is named \$0 and the modules disk is \$MODULES.

Backup

A disk sector copying module will load from the system disk and a reminder to insert the proper disk in the proper drives will be displayed. Also a chance to abort this task by entering "A" will be given. The user may also abort the task by entering zero for a disk number. If two valid disk numbers are entered, the sector copying procedure will begin and exit to desktop when finished.

NOTE: This module will only work with two disk drives and is not buffered. It simply reads a sector from the source disk and writes it to the copy disk. Both disk must already be formatted. REMEMBER, that sector copiers destroy the previous contents of the copy disk and produce an exact clone of the source disk. It will not unfracture the files from the source.

Special

Load & Run

After selecting this option you will be prompted for a program module name. Enter the device and filename. Example: DSK1.MODULE1 . To run, the module must be an ED/ASSM option #3 type program and have been set up to auto run (See instructions for creating an auto run module). Upon exiting the module you will return to desktop. To abort this task without running a program module, press the ENTER key without entering a filename. If the selected module produces an error during the loading process an error window will be displayed. Press the fire button to remove the error window. If the program was selected from the auto catalog it will load and run automatically.

Link & Run

The Link and Run option will load multiple option #3 files, as you would with the ED/ASSM option #3 loader. The name entered at the prompt is the name of a text file which contains a list of modules to be loaded. The last module to be loaded must be an auto run module. As explained in the ED/ASSM manual pg. 306, if an error occurs during the loading process the program will not run and the memory pointers will be reset. The most common errors would be duplicate definitions, a reference to a non-existing label and a lack of memory space to load a module. If this occurs, an error window will display the error number and upon pressing the fire button, the memory pointers will be reset as designated by cSHELL99. Remember, all functions and labels placed on the REF/DEF table are available to user created modules by using a REF or extern in your module. Familiarize yourself with the supplied list to avoid duplicate definition errors.

NOTE: Both the Link & Run and Batch Run options load the linker utility from the system disk before the text list is accessed. If you have chosen to run as a single disk system, a pause window will be displayed after the linker has been loaded. Swap disks at this time and press the fire button to continue with the loading process.

Batch Run

The Batch Run option is similar to the Link & Run option with the exception that each module named in the text file list must be an auto run program module (ED/ASSM option #3). Each module in the text list will be loaded and executed. Upon completion of each module the memory pointers will be reset and the next program module will load and run (Up to 15 modules total). Upon exiting the last program module in the list, the user will return to the desktop. As with Link & Run, if the text file was selected from the auto catalog, the batch processing will execute automatically.

Program Ldr

The Program Ldr option uses a modified version of Todd Kaplan's CHAIN function to load and run ED/ASSM option #5 (program image) files. Many program image files not created for cSHELL99 will load and run using this loader but will not return to the shell unless they were created from within cSHELL99 using the \$SAVE utility (See Creating Program Image Files). If the program was selected from the auto catalog it will load and run immediately. If not, enter the device and filename of the program when prompted. Example: DSK1.NAME . If an error should occur during the loading process an error window will display the error number. Press the fire button or "Q" to remove the window. NOTE: This loader checks for a possible overwrite of the window data in VDP memory and if needed, will attempt to restore this area from a temporary buffer in low memory. Before the program module executes, the screen data and character definitions table for characters 0-239 will be stored in VDP memory starting at >1000. Upon return from the user program, the desktop screen and character definitions will be swapped back into their normal addresses allowing the user module to change characters in text or graphics mode without having to save the existing values. (See VDP Memory Map)

Editor

As of now this option will load TK WRITER from the system disk. You must copy EDITA1/2 and CHARA1 of TK WRITER and \$MEDIT from the \$MODULES disk (TKW files not supplied). If you create your own editor and wish this option to load it, the program must be an option #3 program set to auto run and must be named \$MEDIT on the cSHELL99 system disk.

c99 Compiler

Copy files C99C, C99D and C99E to the cSHELL99 system disk (Files not supplied). The c99 V4.0 compiler will exit to the TI title screen. I hope to find a method to return or at least chain the compiler to the system.

Exit cSHELL99

Selecting this option will terminate cSHELL99 and reset the computer to the TI title screen.

Loader Notes:

All the loaders contained in the desktop "Special" window (Load & Run, Link & Run, Batch Run and Program Ldr.) will save the desktop screen and character definitions table for characters 0-239 into a temporary buffer in VDP memory starting at address >1000 to >1B40 (see VDP Memory Map). Upon return from the user created program The desktop screen and character definitions will automatically be rewritten from this buffer thus allowing a user to redefine characters and change the screen in both text and graphics mode without having to save the data to memory or a disk file. Also, all necessary system globals needed for desktop operation and current memory configuration are saved before and restored after each loader selection. The user created program must not alter this area. If you have need of this area of VDP memory save it to a disk file and restore it using binfil() before exiting your module (see cSHELL99 System Functions).

You may create a program image module (ED/ASSM option #5) that will load and execute in low memory (Almost all of the desktop modules are located in low memory). If you do, the module must not exceed 4096 bytes in size or some of the window data in VDP memory may be overwritten. Also usage of the c99 stack should be minimal to avoid an overlap between your module and the c99 stack (See Low Memory Usage). A program image module that resides in high memory may use the entire available memory and will not destroy any window data because the Program Ldr. loader will restore any overwritten data from a buffer in low memory. Auto run modules (ED/ASSM option #3) may also load into low memory and have no effect on the window data. Again, use the c99 stack with caution to avoid any overlapping of program and stack.

MODULE AUTO RUNNING

The c99 compiler converts "C" source code files into assembly language source files which are assembled with the assembler thus creating Ed/Assm option #3 programs. To enable a module to auto run when it is loaded is a very simple process. All that is needed is to have the starting function's name be placed on the REF/DEF table by the use of the entry statement and also place it after an END assembler statement. The following example will demonstrate this procedure.

```
/* An example of an auto run program module */

entry autol; /* Place autol on the REF/DEF table */

autol() /* This is the starting function of the module */
{
    int x;
    x=0;
    putchar(12); /* Clear the screen */
    while(x<10){
        puts("This is a test .....\\n");
        x++;
    }
    func2(); /* Execute func2 */

} /* Exits module and returns to calling function */
/* DO NOT use exit() or abort() */

func2() /* Just another function in the module */
{
    puts("This is another function");
}

#asm
    END AUTO1 Auto run feature. Note the capital letters.
#endasm
```

In the above example, when this program module is loaded it will start executing autol() and return to the calling function upon completion of autol(). The assembler END statement with the starting name is the last statement in the source file. When compiled, the object file (assembly source) will contain two END statements with the above one first followed by one supplied by the compiler. The assembly process will stop at the first one without causing any errors.

When creating auto run program modules to be used with cSHELL99 the function main() can not be used as a function name because it is already defined as the start of cSHELL99. Also any labels placed on the REF/DEF table as a part of cSHELL99, CSUP, CFIO or ED/ASSM utilities can not be used as a starting function name. Function names that are not made external (func2() above) may be called anything except main() without producing a duplicate definition error.

MODULE AUTO RUNNING/ANOTHER METHOD

Here is another method of running a module without altering the module itself. This method can be used when using the LINK & RUN option or any linker that loads option #3 modules together from a text supplied list. Using the above example again, all the code would remain the same except the #asm, END AUTO1, #endasm would not be included. Instead, create a function module to branch to the starting function and make this function module auto run.

```
                /* Indirect auto running */

entry bootit;          /* Place bootit on REF/DEF table */
extern autol();         /* Access autol from REF/DEF table */

bootit()
{
    autol();
}

#asm

    END BOOTIT          Auto runs bootit which runs autol

#endasm
```

This method can be used to auto start a c99 main() module by replacing autol() with start(). REMEMBER, The auto running module must be the last module to be loaded. DO NOT attempt to start a main() function from the cSHELL99 environment. This method may be used with text list linkers such as CLOADV3A.

LINKING LOADER

The Linking Loader which is included in the utility programs in the Editor/Assembler can load both compressed and uncompressed tagged object code programs (Ed/Assm option #3 files). As explained on page 305 of the Ed/Assm manual, the loader attempts to load relocatable programs in high memory (>A000 - >FFD7) first. If there is insufficient space there, it will try to load programs between the utilities and the REF/DEF table in low memory (about >2676 - >3F37). This type of loader lends itself to creating very powerful programs if used correctly and to my knowledge does not exist in any other micro in the TI's price range. It allows separately created program modules to be linked together and run as one program, sharing the individual module routines and data if desired. The linking loader can also be used within a running program to load and execute program modules if the proper procedure is adhered to. Besides the data on the REF/DEF table, the loader uses these four memory locations in the loading process.

- >2024 First free memory address in high memory
- >2026 Last free memory address in high memory
- >2028 First free memory address in low memory
- >202A Last free memory address in low memory

When cSHELL99 begins to run it copies the values of the above memory addresses to four integer variables. This saves the pointers to the memory area and REF/DEF table used by cSHELL99. To load an Ed/Assm option #3 program file, cSHELL99 uses the function loadit() which takes a device and filename string which is passed to it, creates a PAB, and executes the assembly language instruction LOADER. The program module is loaded and executed if set up to auto run (see Module Auto Run) and upon exiting the module will return to the calling function. Upon return to cSHELL99 the four memory addresses above are reset from the four integer variables that were saved at the startup of cSHELL99. In effect, what this process is doing is to create an overlay area of memory both with the memory available for the module to be loaded and the REF/DEF table information put on as a module loads. By resetting these locations, the loader isn't aware that a program module has been loaded and executed and will re-use the same memory over and over again. If these four addresses were not reset, the memory available for modules would be decreased with the loading of each module and double definition errors would occur with the REF/DEF table if a DEF/entry of a same label name from a previously loaded module were to be loaded again. This process of loading and running other program modules from within a running program may be nested up to a reasonable degree. If you wish to use a module that was loaded and run by cSHELL99 as a shell to run other modules repeat the saving and resetting of the memory pointers described above and use loadit(filename) to load and run the module. In the following example the necessary steps are illustrated.

```

/* Example of a shell program within the cSHELL99 shell */

extern loadit();          /* Loadit must be referenced to be used */
extern savptr(), rtnptr(); /* Memory management functions */
entry shell2;             /* Starting function of this program */

shell2()                  /* Starting function */
{
    savptr();              /* Save the memory configuration of this module */
    loadit("DSK#.MODULE1"); /* Loads and runs module1 from dsk# */
                           /* Loading changes memory pointers */
    rtnptr();              /* Resets memory pointers */
    loadit("DSK#.MODULE2"); /* Loads and runs module2 from dsk# */
    rtnptr();              /* Resets memory pointers */
    loadit("DSK#.MODULE1"); /* module1 again */
    rtnptr();              /* Not needed if last module before return */
                           /* to desktop. Pointers reset by cSHELL99 */

}                          /* returns to calling function (cSHELL99-desktop) */

/* DO NOT use exit() or abort() to return to desktop */
#asm
    END SHELL2           Auto run feature
#endasm

```

In addition to loading and running ED/ASSM option #3 relocatable modules, the loader will load and run option #3 modules that start at an absolute address. In this case, the memory pointers will not be updated except for the last free address in low memory, which points to the bottom of the REF/DEF table. This enables any entry label to be placed on the REF/DEF table in the usual manner.

CREATING PROGRAM IMAGE FILES

It is possible to create binary program files (ED/ASSM option #5) from Load & Run modules (ED/ASSM option #3) by using the \$SAVE utility on the system disk. \$Save is a relocatable utility that performs the same processing as the TI ED/ASSM "SAVE" utility but from within the cSHELL99 environment. To create the program image file, install the prepared file/s (see examples 1 & 2 following) then install \$SAVE from the system disk. \$Save will write part of the VDP window environment to the system disk to allow for larger programs then place a window on the screen prompting for a device and file name. Example DSK#.NAME . If you are using a single disk system, swap disks at this time. Upon pressing ENTER the file/s will be saved in a binary format which can be loaded with the Program Loader under the "Special" window. If you make a mistake or wish to save the module again, follow the prompts given you. If an error is produced in installing a module, an error window will be displayed showing the I/O error. If any error is produced, re-install the last library or choose "Remove" under the "System" window to remove all installed libraries and repeat the process. When \$Save has been completed it will automatically remove all installed libraries and return to cSHELL99.

To save a module/s as a program image file, \$Save requires that three labels be placed on the REF/DEF table by the installed module/s. (SLOAD, SFIRST, and SLAST see examples.) You may save either a single module as in example #1 or modules and libraries as in example #2.

NOTE: No module installed for this purpose should be set to auto run. Also before returning to cSHELL99, the system global "modnum" should be set to zero and all installed libraries should be removed (see examples).

Example #1 Single Module

```
extern jwait(),putbox(),clbox(),endbox(),bxtext(),modnum,single,llow;

/* The extern allows access to the above labels on the REF/DEF table */

entry sload,sfirst,slast;      /* place labels on REF/DEF table */

#define query 7                /* equates the word query with window #7 */

/* Place the following after any extern, entry or #define but before any
   function or memory allocation command */

#asm
SLOAD
SFIRST LWPI >8300      Load c99 workspace. Note capitals..
      B @MSG          Branch to start of module.
#endasm
```

```

msg()                                /* starting function */
{
    putbox(query,10,10);             /* place window #7 at row 10, col 10 */
    btext(query,2,1,"This is a message",1); /* place message in window & */
    jwait();                         /* wait for button press or "Q" */
    clbox(query,0,0);                /* clear out window 7 */
    endbox(query);                   /* remove window 7 from screen */

    /* the following code removes installed libraries and sets modnum to 0 */

    modnum=0;                        /* set modnum to zero */
    /* removing installed libraries by resetting values in a memory
       function which will execute upon return to cSHELL99 */
    #asm
        LI 0,LLOW
        INCT 0
        LI 1,SINGLE
        AI 1,4
        LI 2,4
LP1    MOV *0+,*1+
        DEC 2
        JNE LP1
    #endasm

}                                     /* End of program. Return to cSHELL99 */
#asm
SLAST END                            last memory location in module
#endasm

```

You would compile and assemble the above example then install the option #3 module produced. After installing the module, install \$SAVE and follow the prompts. If you wish to abort this process, press ENTER without supplying a device and filename.

Example #2 Multiple Modules

This example will require SLAST to be placed in a module different than the module containing SLOAD and SFIRST to allow for other libraries to be included in the saving process. The module \$LAST on the system disk may be used to supply the SLAST label.

```

extern printf(),modnum,llow,single; /* access labels from REF/DEF table */

entry sload,sfirst;                /* place sload & sfirst on REF/DEF table */

#asm
SLOAD
SFIRST LWPI >8300    load c99 workspace
        B    @START2 branch to starting function
#endasm
char strg[33]=("This is a message on the screen");

```



```

start2()                                /* starting function */
{
    putchar(12);                        /* clear screen */
    locate(10,1);                      /* position cursor */
    printf("%s",strg);                 /* print message */
    getchar();                         /* wait for keypress */

    /* the following code removes installed libraries and sets modnum to 0 */

    modnum=0;                          /* set modnum to zero */
    /* removing installed libraries by resetting values in a memory
       function which will execute upon return to cSHELL99 */
    #asm
        LI 0,LLOW
        INCT 0
        LI 1,SINGLE
        AI 1,4
        LI 2,4
LP1    MOV *0+,*1+
        DEC 2
        JNE LP1
    #endasm
}                                         /* End of program. Return to cSHELL99 */

```

The above example would be compiled and assembled with the resultant option # 3 module being installed first. All needed libraries not contained in cSHELL99 would then be installed. After the libraries are installed (printf in this example), install the module \$LAST from the system disk. Finally, install \$SAVE from the system disk and follow the prompts. On return from \$SAVE, all installed modules will be removed automatically.

If you use the name "start2" as the starting function of your program module as above, you may use the module \$FIRST on the system disk to supply the SLOAD and SFIRST labels. In this case, eliminate the following in the above example.

```

    entry sload,sfirst;
    #asm
        SLOAD
        SFIRST LWPI >8300
            B    @START2
    #endasm

```

And add an entry for start2. Example: entry start2;
 This time install \$FIRST as the first module followed by the program module, needed libraries, \$LAST and \$SAVE.

cSHELL99 SYSTEM FUNCTIONS

The following is a description of cSHELL99 functions and globals that may be used in user created modules. To access the functions and globals an extern statement must be contained in the user created module.

upper(strg); Upper will replace all lower case characters in the string "strg" to upper case. All non-lower case characters will remain unchanged.

jwait(); Jwait is a pause that will end with the pressing of the fire button on the joystick or "Q" on the keyboard.

strncpy(strg1,strg2); Strncpy will copy strg2 into strg1.

x=joyst2(); The joyst2() function will read and return the position of joystick #1 as a number from 1 thru 18 (description below). It also detects the arrow keys and "Q" on the keyboard and returns their value as a corresponding position of the joystick.

Example:

Down arrow	= 4
Left arrow	= 8
Right arrow	= 12
Up arrow	= 16
"Q"	= 9

In addition, joyst2() will detect a press to key #5 and load and run a specified relocatable screen dump program module created to auto run (see SCREEN DUMP UTILITY).

Joystick Return Values

*	2	4	6	8	NA	12	14	16	18
				10					
*	↙	↓	↘	←		→	↖	↑	↗
%	1	3	5	7	9	11	13	15	17
				BO					
* = Without Button Pressed									
% = With Fire Button Pressed									
NA = No Action									
BO = Button Only									

mouse Mouse, a system global, contains the return value of the last call to the `joyst2()` function and may be a value 1 - 18 (for meaning see `joyst2()`).

n=mmove; The `mmove` function calls `joyst2()` to move the pointer around the screen within the boundaries set by the system globals `ctop`, `cbot`, `cleft`, and `cright`. When the fire button or "Q" is pressed, the VDP screen address (0-959/0-767) of the present pointer position is returned. The position of the pointer may also be found in `crow` (row 1-24) and `ccol` (column 1 - 40/32). Before calling `mmove()` the first time, set `crow` and `ccol` to the screen position you wish the pointer to appear in and execute `getvdp()` and `putvdp(1)` to save the screen character under the pointer and place the pointer on the screen.

Acceptable Values

<code>ctop</code>	1 - 24	Upper line boundary for pointer.
<code>cbot</code>	1 - 24	Lower line boundary for pointer.
<code>cleft</code>	1 - 40/32	left column boundary for pointer.
<code>cright</code>	1 - 40/32	right column boundary for pointer.

NOTE: Because `mmove()` calls `joyst2()`, the screen dump module is made available by pressing key #5 on the keyboard (see SCREEN DUMP MODULE). To disable the loading of this module in your program module, set the system global "mkeys" to one and upon exiting your program set `mkeys` to zero.

putvdp(asc); `Putvdp` will place the character passed (`asc`) on the screen at the location contained in the system globals `crow` and `ccol`.
Example: `putvdp(1)` will place the pointer on the screen.

getvdp(); This function gets the ascii of the character at the screen location determined by `crow` and `ccol` and places the value in the system global "schar".

loadit(strg); `Loadit` creates a PAB for the filename passed (`strg`) and branches to TI's relocatable program loader (LOADER) which loads and links ED/ASSM option #3 program files (normal c99 language program modules). The string passed is the device and filename of the module to be loaded and may be in either upper or lower case. Example : `DSK1.MODNAME` . If the module was created to auto run, upon exiting the program module the return will be to the calling function in `cSHELL99` (see MODULE AUTO RUNNING). `Loadit` is used in `cSHELL99` by the Load and Run, Link and Run, Batch Run and Install options of the desktop. Read the documentation for each option to become familiar with the different uses of `loadit()`. If a module is not successfully loaded, `loadit` will display an I/O error window with the error number displayed. Upon pressing "Q" or the fire button you will return to desktop.

binfil (op, fname, vdpadd, bytes); Binfil creates a PAB for filename (fname) and loads/saves the vdp buffer area (vdpadd) to/from the filename for (bytes) number of bytes.

op = 1 Read a file into vdp buffer.
op = 0 Save the vdp buffer to a file.
fname = Filename of file for save/read (ex. DSK#.NAME).
vdpadd = Decimal starting address of VDP buffer.
bytes = Number of bytes to save/read.

Binfil is a very powerful and versatile function and can be used for a multitude of purposes such as loading/saving screens, character definitions, data, program segments, etc..

Example: binfil(0,"DSK1.SCREEN",0,960); will save a text mode screen to the file "SCREEN" on disk drive #1. To reload the same screen, change the op code to "1". (binfil(1,"DSK1.SCREEN",0,960); . If an error occurs, a window will be displayed on the screen showing the error number. Press "Q" or fire button to remove the error window.

aatoi (x, strg); Aatoi is atoi() renamed to allow for the usage of the printf() function. It takes a string (strg) consisting of numbers and sets the integer (x) to the value represented by the string. Example: If strg="12345", then x would equal the number 12345.

iitod (x, strg, nbytes); Iitod is itod() renamed to allow for the usage of the printf() function. It creates a string (strg) representing the value of the integer (x) nbytes in size. The number of bytes must include a byte for the null string terminator and a byte for the sign of the number.

cSHELL99 WINDOW FUNCTIONS

The following window functions are a customized subset of a previous library created by myself and latter enhanced greatly by Tom Wible of Sterling, Va.. There are many changes in the windowing system such as an increase in the amount of VDP memory available for window data and the method of swapping the screen area with a window. Also because of the increase in memory available for windows the number of window that can be created is increased from 20 to 30. Though some of the previous function call will work with this version, it is recommended that you check window using modules with the following syntax to ensure compatibility. The following functions are contained in cSHELL99 and are available to user programs by using externs in their modules except for makbox() and getbox() which are in \$ADDBOX (a separate library which may be installed). Some of the functions that were in the previous version will be included as add on libraries that can be loaded. All window functions will now work in text and graphics mode in the same manner.

n=makbox(width,lines,strg);

This function will create dialog boxes (windows) in VDP memory and set the appropriate pointers in the array boxdta. To create a window the function must receive the following information: The width of the box in characters, the length of the box in lines and a string containing the actual characters that the window will be comprised of. If you wish the window to be in inverse video, the string containing the window data must be converted using the ivideo(strg) function before calling the makbox() function. Every time this function is executed, it will automatically create the next window number. The first execution is window #1, second is window #2, etc... The window data is stored in VDP memory starting at location 8192 (see VDP MEMORY MAP) and allows for a minimum of 4034 bytes of window data to a maximum of 5558 bytes depending whether an auto processing catalog is active. (see VDP MEMORY MAP). The array boxdta, which contains all pointers to the window and screen data will allow for the creation of up to 30 windows. The amount of bytes a window will occupy can be found by multiplying the width times the length. The function returns the number of the window created or 0 if the window was not created because of a total box data exceeding 5558 bytes or 30 windows being already defined and active.

Example: n=makbox(20,5,string);

Makes a window 20 characters wide and 5 lines long from the specified string. (100 bytes)

n=getbox(filename);

This function, created by Tom Wible of Sterling,VA., will read the D/V 80 file specified and create a window from the lines in the file. It returns the window number if successful, null if not. The file is a text file which contains the window data as you wish it to appear. The width of the window is set by the first line in the file and will be the number of lines in the file in length. Do not include extra blank lines on the end of the file. Also, the text must start at column #1.

Example: n=getbox(dsk1.windowa);


```

*****
* This is a sample box *   -- DSK1.WINDOWA
* in a text file.      *
*****

```

NOTE: If you wish to have the border of the window as a hi-res border such as in cSHELL99, run the module Border with the file before calling this function. (see Companion Modules)

n=clbox(box,start,stop);

This function will turn all characters from window line start to line stop to blanks. Clbox() is used to blank out any part of the window that has been used to display a message allowing the same window to be used repeatedly for different messages or inputs. It must be called before endbox() for windows used for these purposes or the message in the window will become part of the window upon calling endbox(). The function returns a one if successful or a zero if not. Windows that only display the data that they were created with should not be used with this function.

Example: clbox(3,4,7);

This example will clear line 4, 5, 6 and 7 in window #3.

To clear the entire window, except for the border, use zero for the start and stop parameters.

Example: clbox(3,0,0);

This example will clear all of window #3 except for the borders.

n=putbox(box#,line,col);

If the specified window is not already on the screen, putbox() will place the window on the screen with its upper left corner at the screen line and column. The entire window must be able to be contained within the 40/32 columns or a distorted window will be the result. If the window is already on the screen nothing will happen and a null will be returned else a one will be returned.

Example: n=putbox(query,10,10);

This example will place the window "query" on the screen at screen row 10 and screen column 10.

n=endbox(box#);

This function will pop a window off the screen revealing what was underneath it. If the window was not on the screen, nothing will happen and a null will be returned. If successful, a one will be returned. Because the windows directly swap data with the screen below it, the function clbox() should be used on those windows used for varying messages or inputs before calling endbox().

Example: n=endbox(3);

This example will pop window #3 off if it was on the screen.

NOTE: If there are multiple windows on the screen and they overlap, it is the programmer's responsibility to pop the windows off in reverse order.

The top window must be popped off first then the window below it etc...

If the windows do not overlap, they may be popped off in any order.

```
n=menu(window#);
```

This function will enable any window to be used as a menu. By using joystick #1 or the arrow keys on the keyboard a user may move to any line within the window causing the current line to be in inverse video. When the fire button or "Q" key is pressed, the function will return the window line number of the current line at the time of selection. The line number will always start at 2 (to allow for the border) and increase by one to n (bottom line of the window). To move down one line, pull the joystick back or press the down arrow (X). To move up one line, push the joystick forward or press the up arrow (E). To make a selection and return to the calling function, press the fire button or "Q" on the keyboard. If the window is not on the screen a zero will be returned else the window line number will be returned.

Example: n=menu(10);

If on the screen, window #10 will become a menu.

NOTE: When using a window as a menu the window should be on top with no portion under another window. Also this function calls the function joyst2() to access the joystick and arrow keys. As a result the screen dump module that loads when key 5 is pressed is available. If you do not wish to have this module available, set the system global mkeys to a one. Upon exiting your module, reset mkeys to zero. If you do allow for the loading of this utility in your program you must set the memory pointers in your program by using the savptr() function to set the shell to include your program as described in LINKING LOADER/a shell program within a shell program and the global mkeys should not be accessed. Savptr() will only work properly with relocatable modules. In absolute address modules the global FHI should be set to an address just beyond the end of your program if it is in high memory then call savptr().

```
i=video(strg);
```

This function will take a string of standard/inverse characters and toggle the characters between their normal/inverse video equivalent. Once a string has been converted to its inverse video equivalent, the normal string displaying commands such as puts() will not display it properly. However, the graphic function hchar() and wrtlin() from this library may be used to display the inverse video text. If the inverse video text is to be displayed in a window, the function bxttext() can be used. When a string of inverse characters are toggled back to their normal equivalent, it may again be displayed with normal c99 functions.

```
n=bxttext(box#,wrow,wcol,strg,op code);
```

This function will put/get a string inside a window at window row and window column. The parameters passed are window number, window row, window column, a string with data or to accept data, and an operation code (0 gets a string and 1 displays a string). Wrow #1, Wcol #1 will start one line down from the top of the window and one column in from the left to allow for a border. If the specified window is not on the screen, a null will be returned. If the operation is to receive a string, the number of characters

received will be returned. The string input process will terminate with the enter key being pressed or the right border being reached. If the operation is to put a string in a window, The number of characters displayed is returned. Any string that is too long to fit in a window will be truncated at the right border of the window. Example: `n=bxttext(14,2,1,message,1);` In this example, if window #14 is on the screen, the string "message" will be displayed at window row #2 and window col #1.

`n=sngets(row,col,max,strg);`

(Text mode only unless you compute the graphics row & col)

This function will input a string at screen row and screen column for a maximum of max number of bytes. The input process terminates with ENTER or upon reaching max number of bytes. The function also allows for back spacing (`fctn/5`). Returned is the number of characters entered (len). `Sngets()` is not a window oriented function but is included because it is called by `bxttext()` when inputting a string. It also is a very useful function when called directly.

`rdlin(vdpaddr,strg);`

`Rdlin()` is a function used by many of the window functions and is made available because it can be of great use being called directly. It reads a line of characters into a string `strg` directly from the screen starting at `vdpaddr` for "parm2" number of bytes and adds a null at the end. Before calling this function the system global `parm2` must be set to the number of bytes you wish to read and the VDP address `vdpaddr` must be computed. To compute the VDP address of a screen row and column, the following algorithm can be used.

<code>vdpaddr=((row-1)*40)+(col-1);</code>	Text mode
<code>vdpaddr=((row-1)*32)+(col-1);</code>	Graphics mode

`wrtlin(vdpaddr,strg);`

`Wrtlin()` like `rdlin()` also can be of use when called directly. It writes a line of characters to the screen starting at `vdpaddr` from the string `strg` for "parm2" number of bytes. As in `rdlin()`, `parm2` must be set to the number of characters you wish displayed and the VDP screen address must be computed. `Wrtlin()` is useful in displaying strings containing inverse video characters and/or other non-displayable characters.

NOTE: `Wrtlin()` does not use a null to determine the end of the string hence the used of `parm2`. As with all window functions, `parm2` must have an extern in your program module to be accessed.

WINDOW GLOBALS

Boxdta is a global integer array consisting of 120 elements (0-119) in which 30 sets of 4 consecutive elements contain the following information for each dialog box:

Box Element Usage

- 1 This element contains the VDP storage position for the characters that make up the window and is loaded by the makbox() function. Once loaded, it remains constant and may be accessed by the algorithm:
 $n = \text{boxdta}[(\text{boxnumber}-1)*4]$
This is also the position that screen characters which would be under a given window will be swapped to when the window is put on the screen by putbox().
- 2 This element contains the width of the window in characters and is loaded by the makbox() function. Once loaded it remains constant and may be accessed by the algorithm:
 $n = \text{boxdta}[(\text{boxnumber}-1)*4+1]$
- 3 This element contains the length of the window in lines and is loaded by the makbox() function. Once loaded it remains constant and may be accessed by the algorithm:
 $n = \text{boxdta}[(\text{boxnumber}-1)*4+2]$
- 4 This element contains the starting screen position of the window (VDP address) and is loaded by the putbox() funtion. It is updated every time putbox() is used to put a particular window on the screen. When a window is removed from the screen, this element retains the last position of the window. It may be accessed by the algorithm:
 $n = \text{boxdta}[(\text{boxnumber}-1)*4+3]$
This is also the position that the screen characters will be rewritten to when the window is popped of the screen by the endbox() function.

NOTE: In the "C" language all arrays start with the subscript "0". Thus the information for window #1 is in elements 0 - 3, window #2 is in 4 -7

In addition to the boxdta[] array there are five more significant globals used in conjunction with the window functions.

parm2 This global is used primarily with the rdlin() and wrtlin() functions which are accessed from numerous windowing functions. It is set to the width of a specific window and is used in the matrix weaving routines. Since rdlin() and wrtlin() may be used directly, parm2 must be set to the desired width before these functions are called.

boxpos Initialized to 8192, this global points to the VDP storage area for the next window to be created by the makbox() function.

boxptr This global is initialized to 0 and is used by the makbox() function to load the proper elements of the boxdta array. It points to the element #1 entry for the next window to be created.

boxnum Boxnum contains the window number of the last window created by the makbox() function (windows created 1-30). If seven windows were created then boxnum would contain the number 7.

onscrn This global character array contains 30 elements initialized to zero and is accessed by most of the window functions to check whether a particular window is on the screen. If a window is displayed on the screen, its corresponding element [boxnumber-1] will be set to a one otherwise it will contain a zero. Checking onscrn is very useful when creating additional window functions.

cSHELL99 COMPANION MODULES

The following is a list of program modules and c99 libraries on the MODULES disk side A that have been set up for the cSHELL99 environment. Following the module name will be the suggested loader to use and a description of the program.

1. EARTHBAT (Link & Run)

Earthbat is a text file to be used by the Link & Run option under the "Special" desktop window. The GRAPHICS, SPRITES libraries and the auto run program EARTH will be loaded and run a graphics mode demo with multiple sprites and rotating planets. To return to cSHELL99, press the fire button or "Q" key.

2. SDEMOBAT (Link & Run)

Sdemobat is a text file submitted to the Link & Run loader. *not on disk* The SOUND, GRAPHICS and SPRITES libraries and auto run program SDEMO will load and execute a demonstration of the above libraries and return to cSHELL99.

3. \$REFLOOK (Load & Run)

\$Reflook is an auto run utility module which displays the current labels on the REF/DEF table, the hex address that corresponds to the label and address of the entry in the REF/DEF table. Press "Q" or the fire button to page through the table. If key #5 is pressed, the screen dump utility will load allowing you to print the current screen. After the last screen is displayed, you will return to cSHELL99.

4. SLIDE (Load & Run)

Slide is an auto run program that places the "System" window on the screen and slides it from left to right then enters menu mode. Press the fire button or "Q" to slide the window back to the left where it will again enter menu mode. Press the fire button twice to exit menu mode and return to cSHELL99.

5. BATCHTEST (Batch & Run)

NOTE: Install the GRAPHICS, SOUNDS and SPRITES libraries before submitting this batch list to the Batch & Run loader. Batchtest is a text list of auto run program modules. Each module in the list will load and execute in turn. Upon exiting a module, the next module in the list will load and run. After exiting from the last module, you will again be in the desktop.

6. **\$ADDBOX** (Install) or (As a file in a Link & Run list)

\$Addbox is a library which contains two window functions, getbox() and makbox(), which enable a user to add windows to cSHELL99. See cSHELL99 WINDOW FUNCTIONS for further description.

7. **\$ADDTEST** (Load & Run)

NOTE: Before running this module, install \$ADDBOX.

\$Addtest is an auto run module that uses getbox() to create two windows from the files WIND1 and WIND2 and displays the WIND1 window on the screen. Press the fire button and the WIND2 window will be placed on the screen. Press the fire button to exit the module.

8. **\$BORDER and \$BORDER2** (Load & Run)

\$Border/2 are two auto run programs that take a text file set up for creation of a window and replaces the text border with a hi-res border. The text file must start on line one, all the way to the left and contain no extra blank lines. The processed file is saved back to the original filename.

Example:

```
##### In this example the #'s would be replaced by a
# This is a test # hi-res border. $Border produces a single pixel
# window... # line border and $Border2 produces the same borders
##### as the cSHELL99 windows.
```

BRWINDOW

There is a text file on the MODULES disk named WINDOW. As a test, view WINDOW with the View option of the File window, then run \$Border or \$Border2 using DSK#.WINDOW as the input file. Again view WINDOW and you will see the resultant change in the border. The text file may now be used with getbox() to create a window with a hi-res border.

9. **\$CALEN** (Load & Run)

\$Calen is an auto run program module that will display a calendar for any month from 1/84 to 12/99. After the calendar is displayed, you will be given a choice to select another month. If Key #5 is pressed at this time, the screen dump utility will load allowing you to print the current calendar. If you do not select to repeat the calendar process, the program will exit to desktop.

10. **\$PRSET** (Load & Run)

\$Preset is a printer utility for setting Epson compatible printers to different print modes. Written by Clint Pulley, it was modified slightly to auto run and interface with cSHELL99. Press "x" to exit the program.

11. **\$TDUMP** (Load & Run) (Can replace DUMP on the system disk)
(Rename to "DUMP" first")

\$Tdump is a screen dump utility for printers without dot addressable graphics capability. The printer options window will be displayed as in DUMP but the only options available will be Proceed, Screen Section and Abort. Any character with a value less than 32 will be printed as a period and any character greater than 127 will print as an asterisk. The source code will be included to allow a user to add options for their printer. To return to cSHELL99 select Abort and press the fire button or "Q".

12. **SPEECHBAT** (Link & Run)

A short speech demo that says "I am a Texas Instruments computer" and returns to desktop.

13. **\$SAVEBOX** (Load & Run) (Found on System disk side A)

\$Savebox will save the existing cSHELL99 windows and all active user created windows to the system file WIND so that a user need not create often used windows with every rerun of the program. All windows will retain their current window numbers and characteristics. Make sure side A of the System disk is available before pressing ENTER when prompted.

14. **\$REMOVEBOX** (Load & Run) (Found on System disk side A)

\$Removebox will remove all active user created windows from cSHELL99 and reset the window pointers to cSHELL99's original window configuration. If the user created windows were also saved in WIND and you want to remove them, after executing this module, load and run \$Savebox. (source included)

15. **\$LASTBOX** (Load & Run)

\$Lastbox will remove the latest user created window from cSHELL99 and may be rerun to remove all user created windows one by one. It will not remove any of the fourteen original cSHELL99 windows and does not access the system file WIND for any permanent changes. (Source included).

16. **\$WINDOWADD** (Load & Run)

\$Windowadd is a utility to allow a user to add their own windows to cSHELL99 very easily. It uses the getbox() function to create a window from a text file. When prompted type the name of the text file containing the window data. Example: DSK2.WINDOW . If successful, it will display the number of the window created. The newly created window may now be accessed by the returned number in user created modules. To save the window/s permanently use the \$Savebox module as described above. If the window number returned is zero, the window was not created. Note that the VDP window area and auto catalog grow towards each other and under certain conditions one may overwrite a portion of the other. You may use 4034 bytes of

window definitions before any possible overwrite caused by a disk catalog containing 127 filenames will occur. Each filename entry in the auto catalog occupies 12 bytes (see VDP memory map).

17. **\$FW (Load & Run)**

\$FW is a utility loader which will load the Funlwriter environment. A window will appear on the screen and prompt for the disk drive containing Funlweb. Press the appropriate number on the keyboard and the Funlweb environment will be loaded from the file UTIL1. To abort the loading of Funlweb, press zero as the disk number. Note that to return to cSHELL99 from Funlweb you will have to select the Funlweb program loader #3 and enter DSK#.UTIL5 as the filename. Funlweb files are not included with the cSHELL99 package.

18. **\$SAVETEST(Install)**

\$Savetest is a simple demonstration of creating a program image module from within the cSHELL99 environment. Install **\$FIRST**, **\$SAVETEST**, any c99 libraries (not already included in cSHELL99) that you wish, **\$LAST** and **#SAVE** in that order. When prompted, enter a filename to save the program to. Example: DSK#.PROG . After returning to the desktop you may run the created program image module with the Program Ldr. (See CREATING PROGRAM IMAGE FILES).

SCREEN DUMP MODULE

On the cSHELL99 system disk side B is a bit image screen printing utility called DUMP, written for Epson compatible printers with graphics capability. (Source code on MODULES disk side B). It is a relocatable module that will load any time the joyst2() function is called and key #5 is pressed on the keyboard. It is accessible when the pointer is being moved around the screen by the mmove() function and also when a window is on the screen in menu mode. It also may be loaded and run by using the loadit(strg) function. To disable the loading of DUMP in a user created program, set the system global "mkeys" to one and before exiting the module set mkeys to zero. If you wish to allow for the loading of DUMP by pressing key #5 from your Load & Run program, execute the system function savptr() to set the memory pointers to include your module in the shell (ED/ASSM option #3 relocatable modules only).

When DUMP executes it will place a printer options window on the screen for your selection. The options are:

1. Proceed - Begin printing with the specified options. To stop the printing process before completion, press the fire button or "Q". DO NOT press FCTN/4 to stop the process.
2. Set Tab - Set the horizontal character position to start the printing zone on the page. The tab position must allow for enough width left on the line to print the zone selected as determined by the mode selected (elite, pica, single density, double density).
3. Pica - The default mode. Allows for a 480 dot printing zone in single density and 960 dot zone in double density. Each character in text mode is 6 dots wide and in graphics mode 8 dots (pixels) wide.
4. Elite - Allows for a 576 dot printing zone in single density and a 1152 dot zone in double density.
5. Single Density - The default mode. Sets the print zone to a maximum of 480 dot in pica mode and 576 dots in elite mode. This mode prints using every other pin on the print head.
6. Double Density - Sets the print zone to a maximum of 960 dots in pica mode and 1152 dots in elite mode. This mode will print using all the pins on the print head.
7. Screen Section - This option allows the user to select the portion of the screen that will be printed. It may range from one character to the entire screen. The pointer will appear in the upper left corner of the screen. (Some monitors will not show this position in graphics mode). Move the pointer to the upper left corner of the section you want to print and press the fire button once. After the pointer flickers, move the pointer to the lower right corner of the desired section and press the fire button once. The rectangular area between the first and second selection will be printed as

designated by the options selected when you select Proceed. The default print area is the entire screen.

8. Abort - Exits the printing module and returns to the calling function (cSHELL99).

Before each printing the options must be selected or the default values will be in effect. They are Tab=1, Pica mode, Single Density, and the entire screen will be printed.

Note: The screen dump utility may now be used in graphics mode and text mode in the same manner and will adjust itself to print characters of six or eight pixel widths.

An easier explanation of the printing utility is that you may select any part of the screen to be printed at a selected line position in any one of four different sizes.

c99 SOUND LIBRARY

This sound library is used with the c99 compiler by Clint Pulley and was created by Joe Ross.

To use this library with your C programs you must include the file SOUNDLIB in your program. Example: #include "DSK1.SOUNDLIB". You must also load the relocatable library "SOUNDS" when you run your program with the Load and Run option on the Editor Assembler.

This library consist of five sound functions sounda,soundn,sound1,sound2, and sound3. The reason for breaking the sound command into five functions is to save program space by using only the function needed instead of having to pass all the parameters to one larger function.

The frequencies that may be used are the same as in Basic with the highest frequency being 32567 and the lowest being 110. The duration rate for the sound is from 1 - 4250 with each multiple of 18 equal to 1/60th of a second. The value 1000 would enable sound for one second as in Basic with 500 as a half a second etc... Note that a negative duration value can be used with this version. The only difference is that the attenuation (loudness) uses the values of 0 - 15 instead of 0 - 28 as in Basic. A value of 0 would be the loudest sound and 15 would turn off that particular generator. The noise values -1 through -8 are the same as in Basic with the exception that they may also be positive values 1 - 8. All parameters must be passed in each function and must be in the proper order as specified by the function. If you don't want a particular generator to sound, pass a zero value to the frequency or noise and a value of 0 to its respective attenuation. All parameters may be integers or integer variables.

FUNCTIONS IN SOUNDS

This function will produce one tone.

sound1(dur,freq,attn);

This function will produce two tones.

sound2(dur,freq,attn,freq2,attn2);

This function will produce three tones.

sound3(dur,freq,attn,freq2,attn2,freq3,attn3);

This function will produce three tones and a noise.

sounda(dur,freq,attn,freq2,attn2,freq3,attn3,noise,attn4);

This function will produce a noise only.

soundn(dur,noise,attn);

C99 Speech Function

This speech function was designed by Joe Ross to be used with C99 created by Clint Pulley. To use this function with your "C" program you must include the statement `#extern speech()` in your source module. Also you must load "SPEECH", the relocatable function, when you use the ED/ASSM Load and Run option to run your program.

The following is a list of all letters, numbers, words and phrases that can be accessed by a "C" program using the function `say(addr);`. This list is made of the standard words that can be accessed by the speech synthesizer. The address is passed as a decimal integer/variable. Example: `say(8244);` will make the computer say the word "computer".

Address	Phrase	Address	Phrase
-----	-----	-----	-----
18652	- (Negative)	20915	+ (Positive)
20716	. (Point)	5059	0
5129	1	5212	2
5274	3	5351	4
5425	5	5544	6
5608	7	5687	8
5732	9		
5860	A (ay)	5888	A1 (uh)
5908	about	5993	after
6053	again	6151	all
6192	am	6262	an
6316	and	6419	answer
6498	any	21870	are
6567	as	6632	assume
6693	at		
6722	B	6756	back
6799	base	6722	be
6878	between	6983	black
7050	blue	7094	both
7146	bottom	7200	but
7240	buy	7240	by
7240	bye		
7302	C	7385	can
7440	cassette	7495	center
7554	check	7586	choice
7654	clear	7712	color
7764	come	7815	comes
7902	comma	7962	command
8049	complete	8141	completed
8244	computer	8331	connected
8435	console	8508	correct
8578	course	8640	cyan

8707	D	8764	data
8852	decide	8957	device
9062	did	9156	different
9261	diskette	9344	do
9395	does	9450	doing
9534	done	9625	double
9683	down	9746	draw
9832	drawing		
9931	E	9968	each
5687	eight	10019	eighty
10073	eleven	10166	else
10229	end	10342	ends
10413	enter	10479	error
10655	F	10690	fifteen
10781	fifty	10848	figure
10967	find	11038	fine
11099	finish	11156	finished
11223	first	11284	fit
5425	five	5351	for
11326	forty	5351	four
11391	fourteen	11545	fourth
11636	from	11708	front
11755	G	11816	games
11916	get	11962	getting
12088	give	12173	gives
12284	go	12337	goes
12409	going	12502	good
12538	good work	12616	goodbye
12704	got	12753	gray
12829	green	12926	guess
12992	H	13039	had
13113	hand	13183	handheld unit
13317	has	13386	have
13452	head	13541	hear
13594	hello	13681	help
13541	here	13742	higher
13834	hit	13886	home
13961	how	14063	hundred
14167	hurry		
14227	I	14287	I win
14416	if	14450	in
14517	inch	14586	inches
14667	instruction	14781	instructions
14898	is	14970	it
15022	J	15085	joystick
15180	just		

15242	K	15289	key
15337	keyboard	15439	know
15503	L	15568	large
15641	larger	15719	largest
15838	last	15902	learn
15992	left	16050	less
16136	let	16175	like
16234	likes	16341	line
16459	load	16595	long
16701	look	16785	looks
16871	lower		
16947	M	16999	made
17070	magenta	17198	make
17277	me	17341	mean
17413	memory	17516	message
17623	messages	17745	middle
17811	might	17887	module
17986	more	18067	most
18143	move	18237	must
18310	N	18368	name
18483	near	18560	need
18652	negative	18777	next
18853	nice try	5732	nine
19022	ninety	15439	no
19115	not	19162	now
19232	number		
19325	O	19386	of
19475	off	19325	oh
19521	on	5129	one
19595	only	19676	or
19764	order	19850	other
19924	out	19978	over
20070	P	20127	part
20192	partner	20273	parts
20353	period	20453	play
20525	plays	20627	please
20716	point	20808	position
20915	positive	21041	press
21101	print	21162	printer
21241	problem	21344	problems
21486	program	21623	put
21674	putting		
21792	Q		
21870	R	21920	randomly
22098	read (reed)	22465	read1 (red)
22195	ready to start	22341	recorder
22465	red	22529	refer

22625	remember	22735	return
22842	rewind	31800	right
22978	round		
23130	S	23201	said
23279	save	23397	say
23458	says	23547	screen
23643	second	7302	see
23743	sees	23835	set
5608	seven	23888	seventy
23973	shape	24030	shapes
24103	shift	24156	short
24229	shorter	24356	should
24429	side	24520	sides
5544	six	24602	sixty
24688	small	24750	smaller
24817	smallest	24915	so
24983	some	24030	sorry
25126	space	25181	spaces
25292	spell	25395	square
25468	start	25541	step
25591	stop	24983	sum
25635	supposed	25737	supposed to
25844	sure		
25937	T	25995	take
26047	teen	26115	tell
26190	ten	26262	texas instruments
26459	than	26550	that
26646	that is incorrect	26878	that is right
26996	the (thee)	27062	the1 (thuh)
27250	their	27105	then
27250	there	27358	these
27463	they	27552	thing
27663	things	27763	think
27836	third	27921	thirteen
28066	thirty	28126	this
5274	three	28198	threw
28198	through	28265	time
5212	to	28336	together
28447	tone	5212	too
28557	top	28603	try
28687	try again	28818	turn
28878	twelve	28953	twenty
5212	two	29040	type
29118	U	29172	uhoh
29253	under	29341	understand
29487	until	29599	up
29635	upper	29699	use
29769	V	29831	vary
29914	very		

29984	W	30109	wait
30175	want	30241	wants
30384	way	30333	we
30384	weigh	30109	weight
30487	well	30556	were
30652	what	30697	what was that
30837	when	30891	where
30964	which	31012	white
31081	who	31156	why
31249	will	31339	with
5129	won	31403	word
31498	words	31605	work
31676	working	31800	write
31885	X		
31922	Y	31992	yellow
32088	yes	32153	yet
29118	you	32219	you win
32333	your		
32409	Z	5059	zero

LOW MEMORY USAGE BY cSHELL99

<u>Decimal</u>		<u>HEX</u>
8192	-----	>2000
	Ed/Assm Utilities	
9846	-----	>2676
	Low Memory Program Space	
	(Grows upward)	
	v -----	
	^	
	Initial c Stack Position	
	(Grows downward)	
14638	-----	>392E
	^	
	REF/DEF Table	
	(Grows downward)	
16383	-----	>3FFF

VDP RAM USAGE BY C-SHELL99 IN TEXT MODE

Decimal		Hex
0	-----	>0000
	Screen Image Table	
960	-----	>03C0
	Free Space & Sprite Descriptor Block (Experiment)	
1920	-----	>0780
	Sprite Motion Table (Experiment)	
2048	*****	>0800
	Pattern Generator Table (Chars 0-31) (Used by the Desktop screen)	
2304	-----	>0900
	Standard Characters (Chars 32-127)	
3072	-----	>0C00
	Chars 128-135 Window Borders Chars 136-144 (Free)	
3208	-----	>0C8B
	Inverse Video Characters Chars 145-239	
3968	-----	>0F80
	Chars 240-255 (Free)	
4096	*****	>1000
	Used for desktop screen swapping. 960 bytes	
5056	-----	>13C0
	Temporary save of character def. 0-239	
6978	-----	>1B42
	Sound Library VDP Buffer	
7024	-----	>1B70
	CFIO & TCIO PABS & Buffers	
8192	*****	>2000
	:	
	Dialog Boxes Buffer	
v	(Grows upward in memory)	

	:	
	Automode Disk Catalog Storage Buffer	
	(Grows downward in memory)	
13394	*****	>3452
	loadit() & binfil() PAB area	
13442	-----	>3482
	loadit() VDP Input Buffer	
14303	-----	>37DF
	Diskette DSR Blocks	
16383	-----	>3FFF

FILE I/O ERRORS

Error Code	Meaning
<u>Standard I/O</u>	
0	Bad device name.
1	Device is write protected.
2	Bad open attribute (incorrect file type, incorrect record length, incorrect I/O mode or no records in a relative file).
3	Illegal operation.
4	Out of table or buffer space on the device.
5	Attempt to read past the end of file.
6	Device error.
7	File error (non-existing file opened in INPUT mode).
<u>Loader Errors</u>	
8	Memory overflow
9	Not used.
10	Illegal tag.
11	Checksum error.
12	Duplicate definitions.
13	Unresolved reference.
14	Incorrect statement
15	Program not found.

VARIABLE & FUNCTION GLOBALS

The following is a list of all global variables and function names that are on the REF/DEF table during the execution of cSHELL99.

CSUP

C\$IND, C\$DIV, C\$REM, C\$ASR, C\$ASL, C\$EQ, C\$NE, C\$LT, C\$GT, C\$GE, C\$LE, C\$ULT, C\$ULE, C\$UGT, C\$UGE, C\$LNAG, C\$SWCH, GETCH, GETS, PUTCHA, PUTS, LOCATE, POLL, TSCRN, C\$SEND, C\$CLS, C\$FILL, C\$VADR, C\$STRT, S\$SEND, S\$WDTH, C\$CALL, C\$GPLL, C\$CTFL, C\$PGRA, C\$SGRA, C\$HOOK

CFIO

FOPEN, FCLOSE, GETC, PUTC, FGETS, FPUTS, FREAD, FWRITE, FSEEK, FDELETE, FERRC, FEOF, REWIND

EDITOR/ASSEMBLER UTILITIES

VSBW, VMBW, VSBR, VMBR, VWTR, KSCAN, GPLLNK, XMLLNK, DSRLNK, LOADER, SCAN, ULTAB, PAD, GPLWS, SOUND, VDPWA, VDPRD, VDPWD, VDPSTA, GRMWA, GRMRA, GRMRD, GRMWD

cSHELL99

MOUSE, CROW, CCOL, SCHAR, CSPEED, CTOP, CBOT, CLEFT, CRIGHT, JOYST2, GETVDP, PUTVDP, MMOVE, LOADIT, SAVPTR, RTNPTR, FHI, FLOW, LHI, LLOW, SYSAR1, RWERR, START, DELAY, JWAIT, PRINTER, DSK, FNAME, COMFLG, FORG, BACKG, UPPER, STRCPY, PUTBOX, ENDBOX, IVIDEO, BOXNUM, BOXPOS, BOXPTR, MENU, BOXDTA, CLBOX, BXTXT, SNGETS, PARM2, RDLIN, WRTLIN, AATOI, IITOD, BATCH, SYSAR2, REMOVE, BINFIL, DOIT, CINDEX, CATIN, CSTART, RCOUNT, SETVAR, MKEYS, MINDIR, MODNUM, MODULE, LOADST, SWAP1, SWAP2, AUTFLG, SINGLE

c99 STACK & REF/DEF TABLE

The c99 stack area and link loader REF/DEF table both exist at >3FFF and grow downward towards >2000 in low memory. Under normal configurations, it is not possible to use the linking loader from within a c99 program and reference labels from the table if the stack is used. The stack overwrites the table thus wiping out the address links. To overcome this problem I simply moved where the c99 stack resides. The address of the stack pointer is in R14 (workspace register 14). To move it include in your main() function, as the first command, an assembler instruction to load R14 to the address you desire. I moved the stack to >392E (1744 bytes lower in low memory) enough for 218 labels on the REF/DEF table. Of course this move lessens the amount of memory available to the stack but the stack area grows and shrinks with the entry and exit of functions. There is still over 4.6K left for the stack before a crash into the utilities would occur. This setup leaves little room for programs to load in low memory but enables the linking loader to be used within a shell program thus allowing for greater programming power. See LINKING LOADER for a further description of the loading process. An example of moving the c99 stack is as follows:

```
        /* Relocating the c99 stack */

main()
{
    #asm
        LI 14,>392E    c99 doesn't use R before registers.
    #endasm

    int .....;      /* The rest of main() */
    char .....;

    .....
    .....
    .....
}
```

If you are using cSHELL99 this procedure has already been executed and should NOT be repeated. This information has been provided to enable the use of the linking loader from within your own program should you choose to write one from scratch.

NOTE: Using the above assembly instruction in main() may generate an "OUT OF CONTEXT" message as the program is compiled. Just ignore the message because the right assembler code is generated.

cSHELL99 CHARACTER DEFINITIONS

CHAR	PATTERN	DESCRIPTION
1	0078605048040000	Pointer
2	0404040404040404	Left boarder
3	8080808080808080	Right boarder
4	FF80808080808080	Menu divider
5	FF04040404040404	Upper right corner
6	FF00000000000000	Top border
7	00000000000000FF	Bottom border
8	04040404040404FF	Lower right corner
9	8444442414140C04	Bottom corner of flap
10	FF80402010100B04	Top left flap
11	C0C0C0C0C0C0C0C0	Right shadow
12	FFFF000000000000	Bottom shadow
13	C0C0000000000000	Corner shadow
14	FF00FF00FF00FF00	Striped top
15	000000FF00FF00FF	Striped bottom
16	FF80809C9C80B0FF	Left side of disk button
17	FF0404E4E40404FF	Right side of disk button
18	7F7F787070787F7F	Disk icon
19	7F7F7F7F00000000	
20	FFFF38181F3FFF7F	
21	7F7FFFFF00000000	
22	047FFF407078787B	Trash icon
23	78787878787878FF	
24	80F8FF08A8A8A8A8	
25	A8A8A8A8A8A8A8FF	
26	0000047FFF7F407F	Printer icon
27	000000001F24405F	
28	00000000FF8810D0	
29	102040F8FFF808F8	
30	409F00FFFFFFF00FF	
31	808080808080B0FF	Bottom left corner

HI-RES BORDER CHARACTERS

128	FFFFD0FFD0D0D0D0	upper left corner
129	FFFF00FF00000000	Top Line
130	FFFF2FFF2F2F2F2F	upper right corner
131	D0D0D0D0D0D0D0D0	left side
132	2F2F2F2F2F2F2F2F	Right side
133	D0D0D0D0FFD0FFFF	Bottom left
134	00000000FF00FFFF	Bottom line